homework5

April 10, 2023

1 Homework 5

12232509 FAN Qingyuan

Reproduce LeNet-5 by PyTorch, and divide the MNIST dataset into training set, validation set, and test set in a reasonable proportion. Train the LeNet-5, test it on the test set after training, and then compute the accuracy of the test result. In addition, write 20 handwritten digits by yourself and test them, outputting the accuracy of the test.

To reproduce LeNet-5, first we need to import the necessary packages including torch, torchvision, numpy and matplotlib. We also use tensorboard to log the training process.

[1]:
```
import torch
import numpy as np
from torch.utils.tensorboard import SummaryWriter
import math
import matplotlib.pyplot as plt
writer = SummaryWriter()

# use manual seed to reproduce the results
torch.manual_seed(42)
torch.cuda.manual_seed(42)
```

[2]:
```
# create a LeNet-5 by PyTorch using nn.Sequential
class LeNet5Sequential(torch.nn.Module):
    def __init__(self):
        super(LeNet5Sequential, self).__init__()
        self.model = torch.nn.Sequential(
            torch.nn.Conv2d(1, 6, kernel_size=5, padding=2),
            torch.nn.Sigmoid(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),
            torch.nn.Conv2d(6, 16, kernel_size=5),
            torch.nn.Sigmoid(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2),
            torch.nn.Flatten(),
            torch.nn.Linear(16 * 5 * 5, 120),
            torch.nn.Sigmoid(),
            torch.nn.Linear(120, 84),
            torch.nn.Sigmoid(),
```

```
            torch.nn.Linear(84, 10),
        )

    def forward(self, x):
        return self.model(x)
```

2 Load datasets

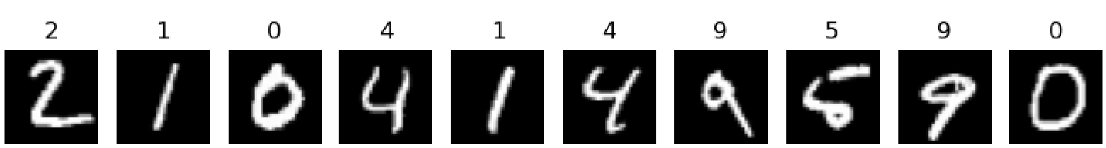
You may skip this chapter if you only want to test the model.

[3]:
```
# load the MNIST dataset
from torchvision import datasets, transforms

transform = transforms.Compose(
    [transforms.ToTensor(), transforms.Normalize((0.1307,), (0.3081,))]
)

train_val_dataset = datasets.MNIST(
    root="./data", train=True, download=True, transform=transform
)
# select 10% data for validation
train_len = int(0.9 * len(train_val_dataset))
val_len = len(train_val_dataset) - train_len
trainset, valset = torch.utils.data.random_split(
    train_val_dataset, [train_len, val_len]
)
testset = datasets.MNIST(root="./data", train=False, download=True,␣
 ↪transform=transform)
```

[4]:
```
# plot some images from the dataset to see what they look like
%matplotlib inline
fig = plt.figure(figsize=(10, 10))
for i in range(1, 11):
    ax = fig.add_subplot(1, 10, i)
    ax.imshow(testset.data[i], cmap='gray')
    ax.set_title(testset.targets[i].item())
    ax.axis('off')
```



[5]:
```python
# train the LeNet-5 model
def train(
    model, trainset, valset, batch_size=64, epochs=10, learning_rate=0.01,␣
 ↪device="cpu"
):
    # define the loss function and the optimizer
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

    # create the data loader
    trainloader = torch.utils.data.DataLoader(
        trainset, batch_size=batch_size, shuffle=True
    )
    valloader = torch.utils.data.DataLoader(
        valset, batch_size=batch_size, shuffle=False
    )

    # train the model
    for epoch in range(epochs):
        model.train()
        for i, (images, labels) in enumerate(trainloader):
            if device == "cuda":
                images = images.to("cuda")
                labels = labels.to("cuda")

            # forward
            outputs = model(images)
            loss = criterion(outputs, labels)

            # backward
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            # log the train loss using tensorboard
            writer.add_scalar("Loss/train", loss.item(), epoch *␣
 ↪len(trainloader) + i)

            # print the loss
            if (i + 1) % 100 == 0:
                print(
                    "Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}".format(
                        epoch + 1, epochs, i + 1, len(trainloader), loss.item()
                    )
                )

        # val the model
```

```
        model.eval()
        with torch.no_grad():
            correct = 0
            total = 0
            for images, labels in valloader:
                if device == "cuda":
                    images = images.to("cuda")
                    labels = labels.to("cuda")
                outputs = model(images)
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()
            # log the train loss using tensorboard
            writer.add_scalar("Accuracy/validation", 100 * correct / total,␣
↪epoch)
            print("Validation accuracy images: {} %".format(100 * correct /␣
↪total))

    # save the last model
    torch.save(model.state_dict(), "model.ckpt")
```

[6]:

```
# train the model
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = LeNet5Sequential().to(device)
testloader = torch.utils.data.DataLoader(testset, batch_size=256, shuffle=False)
train(model, trainset, valset, batch_size=256, epochs=100, learning_rate=0.1,␣
↪device='cuda')
```

```
Epoch [1/100], Step [100/211], Loss: 2.3024
Epoch [1/100], Step [200/211], Loss: 2.3038
Validation accuracy images: 11.1 %
Epoch [2/100], Step [100/211], Loss: 2.3042
Epoch [2/100], Step [200/211], Loss: 2.3042
Validation accuracy images: 11.1 %
Epoch [3/100], Step [100/211], Loss: 2.2980
Epoch [3/100], Step [200/211], Loss: 2.3012
Validation accuracy images: 9.583333333333334 %
Epoch [4/100], Step [100/211], Loss: 2.3052
Epoch [4/100], Step [200/211], Loss: 2.3088
Validation accuracy images: 11.1 %
Epoch [5/100], Step [100/211], Loss: 2.2979
Epoch [5/100], Step [200/211], Loss: 2.2988
Validation accuracy images: 11.1 %
Epoch [6/100], Step [100/211], Loss: 2.3128
Epoch [6/100], Step [200/211], Loss: 2.2994
Validation accuracy images: 11.1 %
Epoch [7/100], Step [100/211], Loss: 2.3051
```

```
Epoch [7/100], Step [200/211], Loss: 2.3001
Validation accuracy images: 10.933333333333334 %
Epoch [8/100], Step [100/211], Loss: 2.3060
Epoch [8/100], Step [200/211], Loss: 2.3203
Validation accuracy images: 11.1 %
Epoch [9/100], Step [100/211], Loss: 2.2998
Epoch [9/100], Step [200/211], Loss: 2.3033
Validation accuracy images: 9.95 %
Epoch [10/100], Step [100/211], Loss: 2.2948
Epoch [10/100], Step [200/211], Loss: 2.2992
Validation accuracy images: 11.1 %
Epoch [11/100], Step [100/211], Loss: 2.2994
Epoch [11/100], Step [200/211], Loss: 2.3003
Validation accuracy images: 10.366666666666667 %
Epoch [12/100], Step [100/211], Loss: 2.3040
Epoch [12/100], Step [200/211], Loss: 2.3037
Validation accuracy images: 11.1 %
Epoch [13/100], Step [100/211], Loss: 2.3054
Epoch [13/100], Step [200/211], Loss: 2.3008
Validation accuracy images: 11.1 %
Epoch [14/100], Step [100/211], Loss: 2.3057
Epoch [14/100], Step [200/211], Loss: 2.3084
Validation accuracy images: 11.1 %
Epoch [15/100], Step [100/211], Loss: 2.2995
Epoch [15/100], Step [200/211], Loss: 2.2941
Validation accuracy images: 10.166666666666666 %
Epoch [16/100], Step [100/211], Loss: 2.3019
Epoch [16/100], Step [200/211], Loss: 2.3102
Validation accuracy images: 10.366666666666667 %
Epoch [17/100], Step [100/211], Loss: 2.3060
Epoch [17/100], Step [200/211], Loss: 2.3034
Validation accuracy images: 11.1 %
Epoch [18/100], Step [100/211], Loss: 2.2971
Epoch [18/100], Step [200/211], Loss: 2.2975
Validation accuracy images: 11.1 %
Epoch [19/100], Step [100/211], Loss: 2.2983
Epoch [19/100], Step [200/211], Loss: 2.2933
Validation accuracy images: 11.1 %
Epoch [20/100], Step [100/211], Loss: 2.3101
Epoch [20/100], Step [200/211], Loss: 2.2918
Validation accuracy images: 11.1 %
Epoch [21/100], Step [100/211], Loss: 2.2947
Epoch [21/100], Step [200/211], Loss: 2.2860
Validation accuracy images: 11.1 %
Epoch [22/100], Step [100/211], Loss: 2.2946
Epoch [22/100], Step [200/211], Loss: 2.2907
Validation accuracy images: 10.166666666666666 %
Epoch [23/100], Step [100/211], Loss: 2.2988
```

```
Epoch [23/100], Step [200/211], Loss: 2.2813
Validation accuracy images: 15.116666666666667 %
Epoch [24/100], Step [100/211], Loss: 2.2808
Epoch [24/100], Step [200/211], Loss: 2.2528
Validation accuracy images: 20.733333333333334 %
Epoch [25/100], Step [100/211], Loss: 2.2186
Epoch [25/100], Step [200/211], Loss: 2.0991
Validation accuracy images: 26.35 %
Epoch [26/100], Step [100/211], Loss: 1.9301
Epoch [26/100], Step [200/211], Loss: 1.7006
Validation accuracy images: 43.25 %
Epoch [27/100], Step [100/211], Loss: 1.5691
Epoch [27/100], Step [200/211], Loss: 1.3539
Validation accuracy images: 55.61666666666667 %
Epoch [28/100], Step [100/211], Loss: 1.2074
Epoch [28/100], Step [200/211], Loss: 1.0504
Validation accuracy images: 66.05 %
Epoch [29/100], Step [100/211], Loss: 0.9908
Epoch [29/100], Step [200/211], Loss: 0.8385
Validation accuracy images: 71.45 %
Epoch [30/100], Step [100/211], Loss: 0.7880
Epoch [30/100], Step [200/211], Loss: 0.6715
Validation accuracy images: 75.06666666666666 %
Epoch [31/100], Step [100/211], Loss: 0.7210
Epoch [31/100], Step [200/211], Loss: 0.6971
Validation accuracy images: 78.75 %
Epoch [32/100], Step [100/211], Loss: 0.6390
Epoch [32/100], Step [200/211], Loss: 0.5159
Validation accuracy images: 82.8 %
Epoch [33/100], Step [100/211], Loss: 0.5845
Epoch [33/100], Step [200/211], Loss: 0.4936
Validation accuracy images: 85.51666666666667 %
Epoch [34/100], Step [100/211], Loss: 0.4939
Epoch [34/100], Step [200/211], Loss: 0.3723
Validation accuracy images: 87.6 %
Epoch [35/100], Step [100/211], Loss: 0.3249
Epoch [35/100], Step [200/211], Loss: 0.3960
Validation accuracy images: 89.33333333333333 %
Epoch [36/100], Step [100/211], Loss: 0.3028
Epoch [36/100], Step [200/211], Loss: 0.2943
Validation accuracy images: 90.46666666666667 %
Epoch [37/100], Step [100/211], Loss: 0.2291
Epoch [37/100], Step [200/211], Loss: 0.2343
Validation accuracy images: 91.61666666666666 %
Epoch [38/100], Step [100/211], Loss: 0.2848
Epoch [38/100], Step [200/211], Loss: 0.2234
Validation accuracy images: 92.33333333333333 %
Epoch [39/100], Step [100/211], Loss: 0.2540
```

```
Epoch [39/100], Step [200/211], Loss: 0.1573
Validation accuracy images: 92.76666666666667 %
Epoch [40/100], Step [100/211], Loss: 0.2336
Epoch [40/100], Step [200/211], Loss: 0.1585
Validation accuracy images: 93.28333333333333 %
Epoch [41/100], Step [100/211], Loss: 0.1614
Epoch [41/100], Step [200/211], Loss: 0.2758
Validation accuracy images: 93.5 %
Epoch [42/100], Step [100/211], Loss: 0.2336
Epoch [42/100], Step [200/211], Loss: 0.1292
Validation accuracy images: 94.01666666666667 %
Epoch [43/100], Step [100/211], Loss: 0.1903
Epoch [43/100], Step [200/211], Loss: 0.1279
Validation accuracy images: 94.33333333333333 %
Epoch [44/100], Step [100/211], Loss: 0.2706
Epoch [44/100], Step [200/211], Loss: 0.2520
Validation accuracy images: 94.73333333333333 %
Epoch [45/100], Step [100/211], Loss: 0.1912
Epoch [45/100], Step [200/211], Loss: 0.1447
Validation accuracy images: 94.91666666666667 %
Epoch [46/100], Step [100/211], Loss: 0.1565
Epoch [46/100], Step [200/211], Loss: 0.1026
Validation accuracy images: 95.08333333333333 %
Epoch [47/100], Step [100/211], Loss: 0.1503
Epoch [47/100], Step [200/211], Loss: 0.1685
Validation accuracy images: 95.23333333333333 %
Epoch [48/100], Step [100/211], Loss: 0.0815
Epoch [48/100], Step [200/211], Loss: 0.1390
Validation accuracy images: 95.38333333333334 %
Epoch [49/100], Step [100/211], Loss: 0.1933
Epoch [49/100], Step [200/211], Loss: 0.1010
Validation accuracy images: 95.65 %
Epoch [50/100], Step [100/211], Loss: 0.1926
Epoch [50/100], Step [200/211], Loss: 0.1254
Validation accuracy images: 95.75 %
Epoch [51/100], Step [100/211], Loss: 0.0887
Epoch [51/100], Step [200/211], Loss: 0.1516
Validation accuracy images: 95.8 %
Epoch [52/100], Step [100/211], Loss: 0.1070
Epoch [52/100], Step [200/211], Loss: 0.1691
Validation accuracy images: 96.01666666666667 %
Epoch [53/100], Step [100/211], Loss: 0.1045
Epoch [53/100], Step [200/211], Loss: 0.0846
Validation accuracy images: 96.08333333333333 %
Epoch [54/100], Step [100/211], Loss: 0.1459
Epoch [54/100], Step [200/211], Loss: 0.1207
Validation accuracy images: 96.23333333333333 %
Epoch [55/100], Step [100/211], Loss: 0.0702
```

```
Epoch [55/100], Step [200/211], Loss: 0.1458
Validation accuracy images: 96.31666666666666 %
Epoch [56/100], Step [100/211], Loss: 0.1184
Epoch [56/100], Step [200/211], Loss: 0.1439
Validation accuracy images: 96.38333333333334 %
Epoch [57/100], Step [100/211], Loss: 0.1153
Epoch [57/100], Step [200/211], Loss: 0.1123
Validation accuracy images: 96.55 %
Epoch [58/100], Step [100/211], Loss: 0.0844
Epoch [58/100], Step [200/211], Loss: 0.0675
Validation accuracy images: 96.61666666666666 %
Epoch [59/100], Step [100/211], Loss: 0.0916
Epoch [59/100], Step [200/211], Loss: 0.0741
Validation accuracy images: 96.68333333333334 %
Epoch [60/100], Step [100/211], Loss: 0.0846
Epoch [60/100], Step [200/211], Loss: 0.1317
Validation accuracy images: 96.8 %
Epoch [61/100], Step [100/211], Loss: 0.0953
Epoch [61/100], Step [200/211], Loss: 0.0905
Validation accuracy images: 96.75 %
Epoch [62/100], Step [100/211], Loss: 0.0801
Epoch [62/100], Step [200/211], Loss: 0.1595
Validation accuracy images: 97.01666666666667 %
Epoch [63/100], Step [100/211], Loss: 0.1227
Epoch [63/100], Step [200/211], Loss: 0.1372
Validation accuracy images: 96.98333333333333 %
Epoch [64/100], Step [100/211], Loss: 0.0974
Epoch [64/100], Step [200/211], Loss: 0.0879
Validation accuracy images: 97.0 %
Epoch [65/100], Step [100/211], Loss: 0.1458
Epoch [65/100], Step [200/211], Loss: 0.0917
Validation accuracy images: 96.88333333333334 %
Epoch [66/100], Step [100/211], Loss: 0.0826
Epoch [66/100], Step [200/211], Loss: 0.1189
Validation accuracy images: 97.2 %
Epoch [67/100], Step [100/211], Loss: 0.0924
Epoch [67/100], Step [200/211], Loss: 0.0705
Validation accuracy images: 97.18333333333334 %
Epoch [68/100], Step [100/211], Loss: 0.0968
Epoch [68/100], Step [200/211], Loss: 0.0817
Validation accuracy images: 97.31666666666666 %
Epoch [69/100], Step [100/211], Loss: 0.0867
Epoch [69/100], Step [200/211], Loss: 0.0568
Validation accuracy images: 97.25 %
Epoch [70/100], Step [100/211], Loss: 0.0783
Epoch [70/100], Step [200/211], Loss: 0.1254
Validation accuracy images: 97.38333333333334 %
Epoch [71/100], Step [100/211], Loss: 0.0946
```

```
Epoch [71/100], Step [200/211], Loss: 0.0772
Validation accuracy images: 97.21666666666667 %
Epoch [72/100], Step [100/211], Loss: 0.0783
Epoch [72/100], Step [200/211], Loss: 0.1150
Validation accuracy images: 97.31666666666666 %
Epoch [73/100], Step [100/211], Loss: 0.0591
Epoch [73/100], Step [200/211], Loss: 0.0873
Validation accuracy images: 97.38333333333334 %
Epoch [74/100], Step [100/211], Loss: 0.1020
Epoch [74/100], Step [200/211], Loss: 0.0591
Validation accuracy images: 97.33333333333333 %
Epoch [75/100], Step [100/211], Loss: 0.1133
Epoch [75/100], Step [200/211], Loss: 0.0785
Validation accuracy images: 97.3 %
Epoch [76/100], Step [100/211], Loss: 0.1029
Epoch [76/100], Step [200/211], Loss: 0.0943
Validation accuracy images: 97.41666666666667 %
Epoch [77/100], Step [100/211], Loss: 0.0548
Epoch [77/100], Step [200/211], Loss: 0.0972
Validation accuracy images: 97.36666666666666 %
Epoch [78/100], Step [100/211], Loss: 0.0552
Epoch [78/100], Step [200/211], Loss: 0.0688
Validation accuracy images: 97.5 %
Epoch [79/100], Step [100/211], Loss: 0.0832
Epoch [79/100], Step [200/211], Loss: 0.1225
Validation accuracy images: 97.45 %
Epoch [80/100], Step [100/211], Loss: 0.0366
Epoch [80/100], Step [200/211], Loss: 0.0666
Validation accuracy images: 97.45 %
Epoch [81/100], Step [100/211], Loss: 0.0927
Epoch [81/100], Step [200/211], Loss: 0.0860
Validation accuracy images: 97.46666666666667 %
Epoch [82/100], Step [100/211], Loss: 0.0965
Epoch [82/100], Step [200/211], Loss: 0.0764
Validation accuracy images: 97.46666666666667 %
Epoch [83/100], Step [100/211], Loss: 0.0688
Epoch [83/100], Step [200/211], Loss: 0.0635
Validation accuracy images: 97.58333333333333 %
Epoch [84/100], Step [100/211], Loss: 0.0650
Epoch [84/100], Step [200/211], Loss: 0.0626
Validation accuracy images: 97.58333333333333 %
Epoch [85/100], Step [100/211], Loss: 0.0869
Epoch [85/100], Step [200/211], Loss: 0.0467
Validation accuracy images: 97.6 %
Epoch [86/100], Step [100/211], Loss: 0.1303
Epoch [86/100], Step [200/211], Loss: 0.1191
Validation accuracy images: 97.53333333333333 %
Epoch [87/100], Step [100/211], Loss: 0.0478
```

```
Epoch [87/100], Step [200/211], Loss: 0.0720
Validation accuracy images: 97.63333333333334 %
Epoch [88/100], Step [100/211], Loss: 0.0732
Epoch [88/100], Step [200/211], Loss: 0.1236
Validation accuracy images: 97.63333333333334 %
Epoch [89/100], Step [100/211], Loss: 0.0646
Epoch [89/100], Step [200/211], Loss: 0.0492
Validation accuracy images: 97.75 %
Epoch [90/100], Step [100/211], Loss: 0.0617
Epoch [90/100], Step [200/211], Loss: 0.0751
Validation accuracy images: 97.56666666666666 %
Epoch [91/100], Step [100/211], Loss: 0.0528
Epoch [91/100], Step [200/211], Loss: 0.1204
Validation accuracy images: 97.7 %
Epoch [92/100], Step [100/211], Loss: 0.0859
Epoch [92/100], Step [200/211], Loss: 0.0728
Validation accuracy images: 97.66666666666667 %
Epoch [93/100], Step [100/211], Loss: 0.0596
Epoch [93/100], Step [200/211], Loss: 0.0907
Validation accuracy images: 97.76666666666667 %
Epoch [94/100], Step [100/211], Loss: 0.0525
Epoch [94/100], Step [200/211], Loss: 0.0551
Validation accuracy images: 97.81666666666666 %
Epoch [95/100], Step [100/211], Loss: 0.0525
Epoch [95/100], Step [200/211], Loss: 0.0585
Validation accuracy images: 97.81666666666666 %
Epoch [96/100], Step [100/211], Loss: 0.0314
Epoch [96/100], Step [200/211], Loss: 0.1125
Validation accuracy images: 97.78333333333333 %
Epoch [97/100], Step [100/211], Loss: 0.0687
Epoch [97/100], Step [200/211], Loss: 0.0547
Validation accuracy images: 97.71666666666667 %
Epoch [98/100], Step [100/211], Loss: 0.0323
Epoch [98/100], Step [200/211], Loss: 0.0813
Validation accuracy images: 97.86666666666666 %
Epoch [99/100], Step [100/211], Loss: 0.0624
Epoch [99/100], Step [200/211], Loss: 0.0976
Validation accuracy images: 97.73333333333333 %
Epoch [100/100], Step [100/211], Loss: 0.0620
Epoch [100/100], Step [200/211], Loss: 0.0813
Validation accuracy images: 97.9 %
```

2.0.1 Use test set to test the model

[7]:
```
# test set accuracy
model.to('cpu')
model.eval()
```

```
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in testloader:
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
    print('Test Accuracy of the model on the test images: {} %'.format(100 *␣
 ↪correct / total))
```

```
Test Accuracy of the model on the test images: 98.15 %
```

[11]:
```
# save model to file
torch.save(model.state_dict(), "model-256batch-98.15.pt")
```

2.1 Verify the model using my own handwritten digits

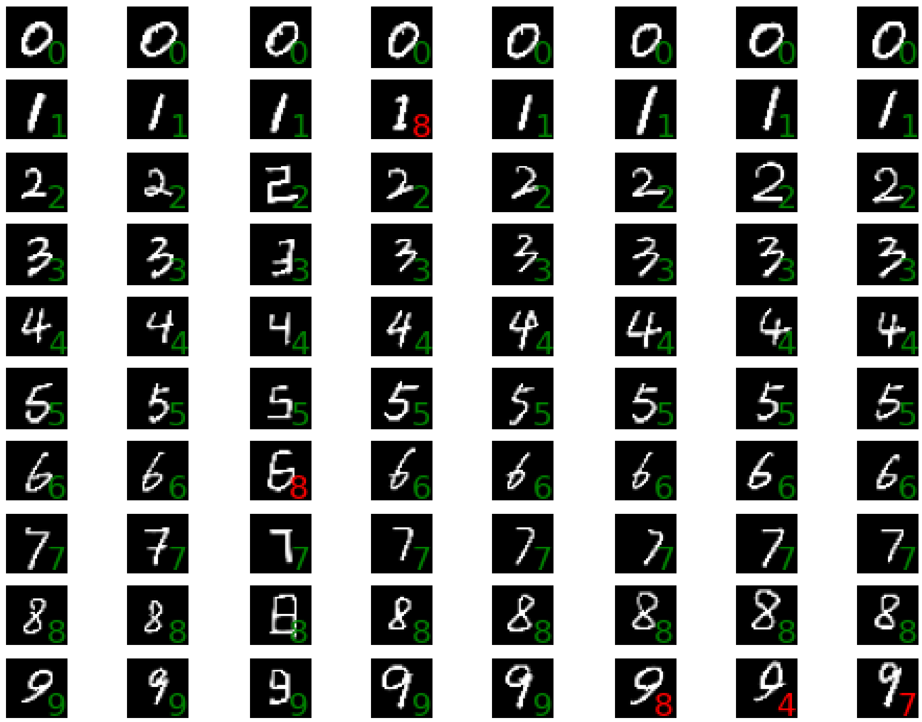
[12]:
```
# load model from file
model = LeNet5Sequential()
model.load_state_dict(torch.load("model-256batch-98.15.pt"))
```

[12]:
```
<All keys matched successfully>
```

[13]:
```
# load npy file
handwrite_test_imgs = np.load('spark_handwrite_test3.npy')
```

[14]:
```
# use the model to predict the handwrite images and plot the prediction results
# put model in evaluation mode and to cpu
model.to('cpu')
model.eval()
corrected_sample_count = 0
with torch.no_grad():
    for i in range(handwrite_test_imgs.shape[0]):
        img = handwrite_test_imgs[i]
        img = torch.from_numpy(img).unsqueeze(0).unsqueeze(0).float()
        output = model(img)
        _, pred = torch.max(output, 1)
        plt.subplot(10, 8, i+1)
        # vertical space between subplots
        # plt.subplots_adjust(hspace=0)
        plt.imshow(handwrite_test_imgs[i], cmap='gray')
        # put the title to the bottom right corner inside the image, use white␣
 ↪color
        # shift down the title a little bit
        if (int(pred.item()) != math.floor(i/8)):
            plt.title(pred.item(), loc='right', color='red', y=-0.2)
```

```
        else:
            plt.title(pred.item(), loc='right', color='green', y=-0.2)
            corrected_sample_count += 1
        plt.axis('off')
```



[15]:
```
# print the accuracy for my own hanwrite datasets
print('Test Accuracy of the model on my own handwritten images: {} %'.
 ↪format(100 * corrected_sample_count / handwrite_test_imgs.shape[0]))
```

```
Test Accuracy of the model on my own handwritten images: 93.75 %
```